

The Atari **ST**^E

Compatible with ST, 1000s of software titles available.

New

- Extended color palette of 4096 colors, from 512
- Hardware support for horizontal and vertical scrolling
- Ready for external GENLOCK
- Stereo 8 bit PCM sound
- Light gun, paddle and new joystick ports.
- 256K ROM from 192K includes
 - Move as well as copy files
 - Rename folders
 - Autoboot GEM applications
 - New file selector
 - Faster desktop
 - Large palette support
 - Fast hard-disk support
 - Folder limitations lifted
 - Memory management improved
 - Keyboard reset

STE Developer Addendum

STE Developer Addendum

This addendum is a set of documents that allows the ST developer to use the new features of the STE. These new features are in the areas of graphics, sound and interface ports.

The STE has a palette of 4096 colors compared to the ST palette of 512 colors. Also the STE has hardware support for vertical and horizontal scrolling. Support has also been added for external GENLOCK.

Sound on the STE has the ST sound as well as 8 bit stereo DMA sound with variable playback frequencies.

The STE also has two new controller ports that allow for new joysticks as well as a light gun and paddle controllers.

Genlock and the STE

The ST (and STE) chip set have the ability to accept external sync. This is controlled by bit 0 at FF820A, as documented in the ST Hardware Specification. This was done to allow the synchronization of the ST video with an external source (a process usually known as GENLOCK). However, in order to do this reliably the system clock must also be phase-locked (or synchronized in some other way) to the input sync signals. No way to do this was provided in the ST, as a result the only GENLOCKS available are internal modifications (usually for the MEGA).

The STE allows this to be done without opening the case. To inject a system clock ground pin three (GPO) on the monitor connector and then inject the clock into pin 4 (mono detect). The internal frequency of this clock is 32.215905 MHz (NTSC) and 32.084988 MHz (PAL). Note: DO NOT SWITCH CLOCK SOURCE WHILE THE SYSTEM IS ACTIVE.

As a result of this GPO is no longer available.

Controllers

FF9200  Fire Buttons

FF9202  Joy 3 Joy 1 Joy 2 Joy 0

Joy sticks.

Four new joystick ports are added. These ports are controlled directly by the 68000. The current state may be sampled at any time by reading the above locations. Joystick 0 and Joystick 2 direction bits are read/write. If written to they will be driven until a read is performed. Similarly, they will not be driven after a read until a write is performed.

FF9210  (X Paddle 0)
 FF9212  (Y Paddle 0)
 FF9214  (X Paddle 1)
 FF9216  (Y Paddle 1)

Paddles.

One pair of paddles can be plugged into Joystick 0 (Paddle 0). A second set can be plugged into Joystick 1 (Paddle 1). The current position of each of the four paddles is reported at these locations. The fire buttons are the same as for the respective joystick. The triggers for the paddles are read as bits one and two of FF9202 (JOY0 Left and Right)

FF9220  (X Position)
 FF9222  (Y Position)

Light Gun / Pen.

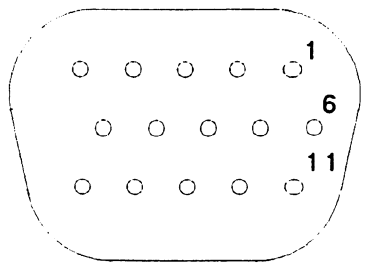
A light gun or pen can be plugged into Joystick 0. The current position that the gun or pen is pointing to is reported by these registers. The position is accurate to within (X direction only):

4 Pixels in 320x200 Mode
 8 Pixels in 640x200 Mode
 16 Pixels in 640x400 Mode

Accurate to 1 pixel in the Y direction in all modes. Accuracies do not account for the quality of the light gun or pen. Note that the X position is given in pixels for 320x200 only. In order to get correct results in 640x200 mode this number needs to be shifted left one bit and in 640x400 mode this number needs to be shifted left two bits.

New Controller Pinout

This pinout is for ports 0 and 1.
Ports 2/3 are on the other DB15 connector.



- 1 UP 0
- 2 DN 0
- 3 LT 0
- 4 RT 0
- 5 PAD 0Y
- 6 FIRE 0
- 7 VCC
- 8 NC
- 9 GND
- 10 FIRE 1
- 11 UP 1
- 12 DN 1
- 13 LT 1
- 14 RT 1
- 15 PAD 0X

Video Modifications

FF8204  (High)

FF8206 

FF8208  (Low)

Video Address Counter.

Now read/write. Allows update of the video refresh address during the frame. The effect is immediate, therefore it should be reloaded carefully (or during blanking) to provide reliable results.

FF820C 

Low byte of the video base address. This register completes the set on ST
Allows positioning screen on word boundaries and thus vertical scrolling.

FF820E 

Offset to next line.

Number of words from end of line to beginning of next line minus one. Allows virtual screen to be wider than physical screen. Acts like an ST when cleared. Cleared at reset.

FF8240 through
FF825E 
Red Green Blue

Color Pallete.

A fourth bit of resolution is added to each color. Note that the least significant bit is added above the old most significant bit to remain compatible with the ST.

FF8264 

Horizontal Bit-wise Scroll.

Delays the start of screen by the specified number of bits.

How to Implement Fine Scrolling on the STE.

The purpose of this document is to describe how to use the capabilities of the STE to achieve bit-wise fine-scrolling and vertical split screens. Horizontal and vertical scrolling are discussed and an example program is provided. Split screen effects are discussed and an example program with multiple independent scrolling regions is provided.

Three new registers are provided to implement fine-scrolling and split screen displays:

1) HSCROLL - This register contains the pixel scroll offset. If it is zero, this is the same as an ordinary ST. If it is non-zero, it indicates which data bits constitute the first pixel from the first word of data. That is, the leftmost displayed pixel is selected from the first data word(s) of a given line by this register.

2) LINEWID - This register indicates the number of extra words of data (beyond that required by an ordinary ST at the same resolution) which represent a single display line. If it is zero, this is the same as an ordinary ST. If it is non-zero, that many additional words of data will constitute a single video line (thus allowing virtual screens wider than the displayed screen). *CAUTION*- In fact, this register contains the word offset which the display processor will add to the video display address to point to the next line. If you are actively scrolling (HSCROLL $\neq$ 0), this register should contain the additional width of a display line *minus one data fetch* (in low resolution one data fetch would be four words, one word for monochrome, etc.).

3) VBASELO - This register contains the low-order byte of the video display base address. It can be altered at any time and will affect the next display processor data fetch. It is recommended that the video display address be altered only during vertical and horizontal blanking or display garbage may result.

These registers, when used in combination, can provide several video effects. In this document we will discuss only fine-scrolling and split-screen displays.

Fine Scrolling:

Many games use horizontal and vertical scrolling techniques to provide virtual playfields which are larger than a single screen. We will first discuss vertical scrolling (line-wise), then horizontal scrolling (pixel-wise) and finally the example program "neowall.s" which combines both.

Vertical Scrolling:

To scroll line-wise, we simply alter the video display address by one line each time we wish to scroll one line. This is done at vertical blank interrupt time by writing to the three eight-bit video display address registers to define a twenty-four-bit pointer into memory.

Naturally, additional data must be available to be displayed. We might imagine this as a tall, skinny screen which we are opening a window onto for the user. The video display address registers define where this window will start.

Horizontal Scrolling:

To scroll horizontally we might also adjust the video display address. If that was all we did, we would find that the screen would jump sideways in sixteen pixel increments. To achieve smooth pixel-wise scrolling we must use the HSCROLL register to select where within each sixteen pixel block we wish to start displaying data to the screen. Finally, we must adjust the LINEWID register to reflect both the fact that each line of video data is wider than a single display line and any display processor fetch incurred by a non-zero value of HSCROLL. All this is done at vertical blank interrupt time. Naturally, additional data must be available to be

displayed. We might imagine this as an extremely wide screen which we are opening a window onto for the user. These registers define where this window will start.

For Example:

The program "neowall.s" reads in nine NEOchrome™ picture files, organizes them into a three by three grid and allows the user to scroll both horizontally and vertically over the images. The heart of this program (the only interesting thing about it actually) is the vertical blank interrupt server. This routine first determines the pixel offset and loads it into HSCROLL. The LINEWID register is now set to indicate that each virtual line is three times longer than the actual display width. If we are actively scrolling, this amount is reduced to reflect the additional four-plane data fetch which will be caused by the scrolling. Finally, the video display address is computed to designate a window onto the grid of pictures. This twenty-four-bit address determines where the upper-left corner of the displayed region begins in memory. Thus, every frame an arbitrary portion of the total image is selected for display. The speed and resolution of this scrolling technique is limited only by the dexterity of the user.

Split Screen:

In many applications it is desirable to subdivide the screen into several independent regions. On the STE you may reload some video registers on a line-by-line basis (using horizontal blanking interrupts) to split the screen vertically into multiple independent regions. A single screen no longer need be a contiguous block of storage, but could be composed of dozens of strips which might reside in memory in any order. The same data could be repeated on one or more display lines. Individual regions might each have their own individual data and scrolling directions.

For Example:

The program "hscroll.s" reads in a NEOchrome™ picture file and duplicates each line of the image. This, combined with the proper use of LINEWID, effectively places two copies of the same picture side-by-side. Next, both vertical and horizontal blanking interrupt vectors are captured and the horizontal blanking interrupt is enabled in counter mode. To prevent flicker caused by keyboard input, the IKBD/MIDI interrupt priority is lowered below that of the HBL interrupt. Note that the program 'main loop' doesn't even call the BIOS to check the keyboard, since the BIOS sets the IPL up and causes flicker by locking out horizontal interrupts - this may cause trouble for programs in the real world. The screen is effectively divided into ten regions which scroll independently of one another. There are two ten-element arrays which contain the base address of each region and its current scroll offset. At vertical blank interrupt time we compute the final display values for each region in advance and store them into a third array. We then initialize the display processor for the first region and request an interrupt every twenty lines (actually every twenty horizontal blankings). During each horizontal interrupt service, we quickly reload the video display address registers and the HSCROLL register. This must be done immediately - before the display processor has time to start the current line or garbage may result. Note that horizontal blank interrupts are triggered by the display processor having finished reading the previous data line. You have approximately 144 machine cycles to reload the HSCROLL and video display registers before they will be used again by the display processor. Finally, the LINEWID register is set, this need only be done before the processor finishes reading the data for the current display line. We then pre-compute the data we will need for the next horizontal interrupt to shave few more cycles off the critical path and exit.

```

1      ;
2      ; HSCROLL.S Horizontal Scrolling Demo
3      ; THE ONE LINE VERSION
4      ;
5      ; Copyright 1988 ATARI CORP.
6      ; Started 9/12/88 .. Rob Zdybel
7      ;
8
9      .text
10     .include atari
569    .list
11
12     ;
13     ; HARDWARE CONSTANTS
14     ;
15     =FFFF820D      vbaselo =      $ffff820d      ; Video Base Address (lo)
16     =FFFF820F      linewid =      $ffff820f      ; Width of a scan-line (Words, minus 1)
17     =FFFF8265      hscroll =      $ffff8265      ; Horizontal scroll count (0 .. 15)
18
19     ;
20     ; SYSTEM CONSTANTS
21     ;
22     =00000070      vblvec  =      $70             ; System VBlank Vector
23     =00000118      ikbdvec =      $118            ; IKBD/MIDI (6850) Vector
24     =00000120      hblvec  =      $120            ; Horizontal Blank Counter (68901) Vector
25
26     ;
27     ; LOCAL CONSTANTS
28     ;
29
30     ;
31     ; System Initialization
32     ;
33     start:
34     00000000 2A4F      move.l  a7,a5
35     00000002 2E7Cxxxxxx move.l  %mystack,a7      ; Get Our Own Local Stack
36     00000008 2A6D0004   move.l  4(a5),a5      ; a5 = basepage address
37     0000000C 202D000C   move.l  TEXTSZ(a5),d0
38     00000010 D0AD0014   add.l   DATASZ(a5),d0
39     00000014 D0AD001C   add.l   BSSSZ(a5),d0
40     00000018 D0BC00000100 add.l   #$100,d0      ; RAM req'd = text+bss+data+BasePageLength
41     0000001E 2800      move.l  d0,d4      ; d4 = RAM req'd
42                                Mshrink a5,d0      ; Return Excess Storage
43                                move.l  d0,-(sp)
44                                move.l  a5,-(sp)
45                                clr.w   -(sp)
46                                Gemdos  $4a,12
47                                move.w  #$4a,-(sp)
48                                trap     #1
49                                .if $c <= 8
50                                addq    #$c,sp
51                                .else
52                                add.w   #$c,sp
53                                .endif
54
55     ;
56     ; Other Initialization
57     ;
58     Super                                ; enter supervisor mode
59     00000030 42A7      clr.l   -(sp)
60     00000032 3F3C0020   move.w  #$20,-(sp)
61     00000036 4E41      trap     #1
62     00000038 5C4F      addq     #6,sp
63     0000003A 2F00      move.l  d0,-(sp)      ; WARNING - Old SSP saved on stack.
64
65     ;
66     Fgetda
67     0000003C 3F3C002F   move.w  #$2f,-(sp)
68     00000040 4E41      trap     #1
69     00000042 544F      .if $2 <= 8
70                                addq    #$2,sp
71                                .else
72                                add.w   #$2,sp
73                                .endif
74     00000044 2840      move.l  d0,a4
75     00000046 D8FC001E   adda    #30,a4      ; a4 = Filename ptr
76
77     Ffirst #neofile,#0
78     0000004A 3F3C0000   move.w  #$0,-(sp)
79     0000004E 2F3Cxxxxxx   move.l  #neofile,-(sp)
80                                Gemdos  $4e,8
81     00000054 3F3C004E   move.w  #$4e,-(sp)
82     00000058 4E41      trap     #1
83     0000005A 504F      .if $8 <= 8
84                                addq    #$8,sp
85                                .else
86                                add.w   #$8,sp
87                                .endif
88
89     tst     d0
90     bmi     abort      ; IF (No NEO files) ABORT
91     Fopen   a4,#0
92     00000062 3F3C0000   move.w  #$0,-(sp)
93     00000066 2F0C      move.l  a4,-(sp)

```

```

00000068 3F3C003D      @      Gemdos $3d,8
0000006C 4E41          @      move.w #$3d,-(sp)
                                @      trap #1
                                @      .if $8 <= 8
0000006E 504F          @      addq #$8,sp
                                @      .else
                                @      add.w #$8,sp
54          -      .endif
55          -      tst d0
56          bmi abort ; IF (Error opening file) ABORT
57
58          move d0,handle
00000070 4A40          @      Fread d0,#32128,#neobuff
00000072 6B00xxxx      @      move.l #neobuff,-(sp)
00000076 33C0xxxxxxx    @      move.l #$7d80,-(sp)
0000007C 2F3Cxxxxxxx    @      move.w d0,-(sp)
00000082 2F3C0007D80    @      Gemdos $3f,12
00000088 3F00          @      move.w #$3f,-(sp)
0000008A 3F3C003F      @      trap #1
0000008E 4E41          @      .if $c <= 8
                                @      addq #$c,sp
                                @      .else
00000090 DEFC000C    @      add.w #$c,sp
59          -      .endif
60          -      tst.l d0
61          bmi abort ; IF (File Read Error) ABORT
00000094 4A80          @      Fclose handle
00000096 6B00xxxx      @      move.w handle,-(sp)
0000009A 3F39xxxxxxx    @      Gemdos $3e,4
000000A0 3F3C003E      @      move.w #$3e,-(sp)
000000A4 4E41          @      trap #1
000000A6 504F          @      .if $4 <= 8
                                @      addq #$4,sp
                                @      .else
62          -      add.w #$4,sp
63          -      .endif
64          -      tst d0
65          -      bmi abort ; IF (Error Closing a file) ABORT
66          -
67          lea neobuff+4,a2
68          lea palette,a0
69          lea oldpal,a1
70          move #15,d0
71          .ploop: move.w (a0),(a1)+ ; save old color palette
72          move.w (a2)+,(a0)+ ; create new color palette
73          dbra d0,.ploop
74          move #160,d0 ; Double each display line
75          lea bigbuff,a0
76          lea neobuff+128,a1
77          move #199,d2
78          .linlp: move #39,d1 ; FOR (200 Lines) DO
79          .dublp: move.l (a1),(a0,d0) ; duplicate line
80          move.l (a1)+,(a0)+
81          dbra d1,.dublp
82          adda d0,a0
83          dbra d2,.linlp
84
85          lea baseaddr,a0
86          lea xoffset,a1
87          lea bigbuff,a2
88          move #9,d0
89          .strlp: move #0,(a1)+ ; FOR (10 Strips) DO Init base and offset
90          move.l a2,(a0)+
91          adda #320*20,a2
92          dbra d0,.strlp
93
94          move.l ikbvec,oldikbd
95          move.l #ikbd,ikbvec ; IPL 5 hack for IKBD/MIDI
96
97          move.l vblvec,oldvbl
98          move.l #vbl,vblvec ; Capture System VBlank Interrupt
99
100         move.l #hbl,hblvec ; Capture HBlank Interrupt
101         bset.b #0,imra
102         bset.b #0,iera ; Enable Hblank
103
104         ;
105         ; Scrolling Demo loop
106         ;
107         wavelp:
0000014A 3F3C0002      @      Bconstat CON ; Keyboard Polling
0000014E 3F3C0001      @      move.w #CON,-(sp)
00000152 4E40          @      Bios 1.4
00000154 504F          @      move.w #$1,-(sp)
                                @      trap #13
                                @      .if $4 <= 8
                                @      addq #$4,sp
                                @      .else
                                @      add.w #$4,sp
                                @      .endif

```

```

109 00000156 4A40          tst     d0
110 00000158 6700xxxx      beq     noexit          ; IF (Keyboard Input Available) THEN
                                Bconin  CON
                                move.w  #CON,-(sp)
                                @
                                Bios 2,4
                                @
                                move.w  #$2,-(sp)
                                @
                                trap     #13
                                @
                                .if $4 <= 8
                                @
                                addq     #$4,sp
                                @
                                .else
                                @
                                add.w    #$4,sp
                                -
                                .endif
111
112 00000168 803C0003      cmp.b   #'C'-64,d0
113 0000016C 6700xxxx      beq     exit          ; CTRL-C ==> EXIT
114
115 00000170 6008          noexit:   bra     wavelp
116
117                      exit:
118                      ;
119                      ;      System Tear-Down
120
120 00000172 8880000FA07      bclr.b  #0,iera
121 00000178 8880000FA13      bclr.b  #0,imra          ; Disable Hblank
122 0000017E 21F9xxxxxxx0000  move.l  oldikbd,ikbvec   ; Restore System IKBD/MIDI Interrupt
123 00000186 21F9xxxxxxx0000  move.l  oldvbl,vblvec    ; Restore System VBlank Interrupt
124
                                Gettime
                                Xbios   $17,2
                                @
                                move.w  #$17,-(sp)
                                @
                                trap     #14
                                @
                                .if $2 <= 8
                                @
                                addq     #$2,sp
                                @
                                .else
                                -
                                add.w    #$2,sp
                                -
                                .endif
125
126 00000196 23C0xxxxxxx    move.l  d0,vbltemp        ; Get IKBD Date/Time
                                Tsettime d0
                                @
                                move     d0,-(sp)
                                @
                                Gemdos   $2d,4
                                @
                                move.w  #$2d,-(sp)
                                @
                                trap     #1
                                @
                                .if $4 <= 8
                                @
                                addq     #$4,sp
                                @
                                .else
                                -
                                add.w    #$4,sp
                                -
                                .endif
127
                                Tsetdate vbltemp        ; Set GEMDOS Time and Date
                                move     vbltemp,-(sp)
                                @
                                Gemdos   $2b,4
                                @
                                move.w  #$2b,-(sp)
                                @
                                trap     #1
                                @
                                .if $4 <= 8
                                @
                                addq     #$4,sp
                                @
                                .else
                                -
                                add.w    #$4,sp
                                -
                                .endif
128
129
130 000001B4 41F9xxxxxxx    lea     oldpal,a0
131 000001B8 43F88240      lea     palette,a1
132 000001BE 303C000F      move    #15,d0
133 000001C2 3208          .unplp: move.w  (a0)+,(a1)+
134 000001C4 51C8FFFC      dbra    d0,.unplp        ; restore old color palette
135
                                abort: User
                                @
                                Gemdos   $20,6
                                @
                                move.w  #$20,-(sp)
                                @
                                trap     #1
                                @
                                .if $6 <= 8
                                @
                                addq     #$6,sp
                                @
                                .else
                                -
                                add.w    #$6,sp
                                -
                                .endif
136
                                Pterm0
                                @
                                clr.w    -(sp)          ; return to GEMDOS
                                @
                                trap     #1
                                @
                                illegal
137
138
139
140
141
142                      ;
143                      ;      VBL      Vertical-Blank Interrupt Server
144                      ;
145                      vbl:
146 000001D6 48E7C0E0      movem.l d0-d1/a0-a2,-(sp)
147
148 000001DA 41F9xxxxxxx    lea     video,a0          ; a0 = Display list (scroll,base)
149 000001E0 43F9xxxxxxx    lea     xoffset,a1        ; a1 = Xoffset list
150 000001E6 45F9xxxxxxx    lea     baseaddr,a2      ; a2 = Base address list
151 000001EC 323C0009      move     #9,d1
152
153                      .reglp:
154 000001F0 3011          move     (a1),d0          ; FOR (10 scrolling regions) DO
155 000001F2 08010000      btst.l  #0,d1          ; d0 = current Xoffset
156 000001F6 6600xxxx      bne     .odd
157 000001FA 5240          addq     #1,d0          ; EVEN --> Increment

```

```

154 000001FC 807C00A0      cmp     #160,d0
155 00000200 6000xxxx      blt     .join
156 00000204 7000          moveq    #0,d0      ; Wrap-up
157 00000206 6000xxxx      bra     .join
158 0000020A 5340      .odd:  subq     #1,d0      ; ODD --> Decrement
159 0000020C 6C00xxxx      bge     .join
160 00000210 303C009F      move     #159,d0      ; Wrap-down
161 00000214 3280      .join:  move     d0,(a1)      ; New Xoffset
162 00000216 E240          asr      #1,d0
163 00000218 C0BC0000FFF8    and.l    #$0fff8,d0      ; d0 = byte offset within line
164 0000021E 009A          add.l    (a2)+,d0      ; d0 = Regions video base
165 00000220 2080          move.l    d0,(a0)
166 00000222 3019          move     (a1)+,d0
167 00000224 C07C000F    and     #$0f,d0      ; d0 = Regions horizontal scroll count
168 00000228 1080          move.b    d0,(a0)
169 0000022A 5888          addq.l    #4,a0
170 0000022C 51C9FFC2      dbra     d1,.reglp
171
172 00000230 41F9xxxxxxx    lea     video,a0
173 00000236 1018          move.b    (a0)+,d0
174 00000238 11C08265      move.b    d0,hscroll
175 0000023C 11D88205      move.b    (a0)+,vcounthi
176 00000240 11D88207      move.b    (a0)+,vcounthi
177 00000244 11D88209      move.b    (a0)+,vcounthi ; Initialize first region
178
179 00000248 323C0050      move     #80,d1      ; Double normal ST line width
180 0000024C 4A00          tst.b     d0
181 0000024E 6700xxxx      beq     .zero      ; IF (non-zero scroll count) Reduce line width
182 00000252 5941          subq     #4,d1
183 00000254 11C1820F      .zero:  move.b    d1,linewid
184
185 00000258 2018          move.l    (a0)+,d0
186 0000025A E198          rol.l    #8,d0
187 0000025C 23C0xxxxxxx    move.l    d0,videodata ; Init next lines data
188 00000262 23C0xxxxxxx    move.l    a0,videopttr ; Init display list ptr
189
190 00000268 11FC0000FA1B    move.b    #0,tbcr
191 0000026E 11FC0014FA21    move.b    #20,tbdr      ; Interrupt every twenty HBlanks
192 00000274 11FC0008FA1B    move.b    #8,tbcr
193
194 0000027A 4CDF0703      movem.l   (sp)+,d0-d1/a0-a2
195 0000027E 4EF9          .dc.w    $4ef9
196 00000280 00000000      oldvbl:  .dc.l    0      ; JMP (Old-Vblank)
197 00000284 4AFC          illegal
198
199 ;
200 ;      IKBD      IKBD/MIDI Interrupt Server
201 ;
202 ikbd:
203 00000286 3F00          move     d0,-(sp)
204
205 00000288 40C0          move     sr,d0
206 0000028A C07CF8FF      and     #$f8ff,d0
207 0000028E 807C0050      or      #$500,d0
208 00000292 46C0          move     d0,sr      ; Set IPL down to 5
209
210 00000294 301F          move     (sp)+,d0
211 00000296 4EF9          .dc.w    $4ef9
212
213 00000298 00000000      oldikbd: .dc.l    0      ; JMP (Old-IKBD)
214 0000029C 4AFC          illegal
215
216 ;
217 ;      HBL      *ONE LINE* Horizontal-Blank Interrupt Server
218 ;
219 hbl:
220 0000029E 48E78080      movem.l   d0/a0,-(sp)      ; (44*28=72)
221
222 000002A2 2039xxxxxxx    move.l    videodata,d0 ; d0 = vcount/scroll (20)
223 000002A8 41F88205      lea     vcounthi,a0      ; a0 = movep base (8)
224 000002AC 11C08265      move.b    d0,hscroll      ; set HScroll (12)
225 000002B0 01C80000      movep.l   d0,(a0)      ; set VideoBase (24)
226                                ; (total = 136+ cycles)
227 000002B4 4A00          tst.b     d0
228 000002B6 6700xxxx      beq     .zero      ; IF (non-zero scroll count) Reduce line width
229 000002BA 11FC004C820F    move.b    #76,linewid
230 000002C0 6000xxxx      bra     .join
231 000002C4 11FC0050820F    .zero:  move.b    #80,linewid
232      .join:
233 000002CA 2079xxxxxxx    move.l    videopttr,a0
234 000002D0 2018          move.l    (a0)+,d0
235 000002D2 E198          rol.l    #8,d0
236 000002D4 23C0xxxxxxx    move.l    d0,videodata ; Init next regions data
237 000002D8 23C0xxxxxxx    move.l    a0,videopttr
238
239 000002E0 4CDF0101      movem.l   (sp)+,d0/a0
240 000002E4 00B80000FA0F    bclr.b    #0,isra      ; Clear In-Service bit
241 000002EA 4E73          rte
242
243 ;
244 ;      DATA STORAGE

```

```

245      ;
246      000002EC      .data
247      neofile:      .dc.b    "*.neo",0      ; NEO filename search string
248      00000000 2A2E6E656F00
249      .even
250
251
252      ;
253      ;      RANDOM DATA STORAGE
254      ;
255      00000006      .bss
256
257      oldpal:      .ds.l    16      ; Original color palette
258      00000000 =00000010
259      handle:      .ds.w    1      ; Active Handle
260      00000040 =00000001
261
262      baseaddr:      .ds.l    10      ; Image Base address for each strip
263      00000042 =0000000A
264      xoffset:      .ds.w    10      ; Pixel-offset for each strip
265      0000006A =0000000A
266      video:      .ds.l    10      ; HScroll and Video Base address for each strip
267      0000007E =0000000A
268      videoptr:      .ds.l    1      ; Display list ptr
269      000000A6 =00000001
270      videodata:      .ds.l    1      ; Next regions display info
271      000000AA =00000001
272
273      neobuff:      .ds.b    32128      ; NEO-Image Buffer
274      000000AE =00007080
275      bigbuff:      .ds.b    2*32000      ; Mega-Image Buffer
276      00007E2E =0000FA00
277
278      vbltemp:      .ds.l    1      ; Vblank Temporary Storage
279      0001782E =00000001
280
281      .ds.l    256      ; (stack body)
282      mystack:      .ds.l    1      ; Local Stack Storage
283      00017C32 =00000001
284
285      .end

```

Symbol Table

.dublp	000000E2	t	dtr	00000010	ea	tacr	FFFFFA19	ea
.join	00000214	t	end_os	000004FA	ea	tadr	FFFFFA1F	ea
.join	000002CA	t	etv_critic	00000404	ea	tbcrr	FFFFFA18	ea
.linlp	0000000E	t	etv_term	00000408	ea	tbdrr	FFFFFA21	ea
.odd	0000020A	t	etv_timer	00000400	ea	tcddrr	FFFFFA1D	ea
.ploop	000000C2	t	etv_xtra	0000040C	ea	tcdr	FFFFFA23	ea
.reglp	000001F0	t	exec_os	000004FE	ea	tddrr	FFFFFA25	ea
.stripl	00000108	t	exit	00000172	t	themd	0000048E	ea
.unplp	000001C2	t	fifo	FFFF8506	ea	trkreg	00000082	ea
.zero	00000254	t	flock	0000043E	ea	trpl4ret	00000486	ea
.zero	000002C4	t	giaamp	00000008	ea	tsr	FFFFFA2D	ea
AUX	00000001	ea	gibamp	00000009	ea	ucr	FFFFFA29	ea
BBASE	00000018	a	gicamp	0000000A	ea	udr	FFFFFA2F	ea
BLEN	0000001C	a	gicrnulp	0000000C	ea	vbasehi	FFFF8201	ea
BPSZ	00000100	ea	gifienvlp	00000008	ea	vbaselo	FFFF8200	ea
B5IZE	0000000A	a	gimixer	00000007	ea	vbasemid	FFFF8203	ea
B5SSZ	0000001C	ea	ginoise	00000006	ea	vbl	000001D6	t
CMDLINE	00000000	a	giporta	0000000E	ea	vblsem	00000452	ea
CON	00000002	ea	giportb	0000000F	ea	vbltemp	0001782E	b
CR	0000000D	ea	giread	FFFF8800	ea	vblvec	00000070	ea
CURS_BLINK	00000002	ea	giselect	FFFF8800	ea	vcounthi	FFFF8205	ea
CURS_GETRATE	00000005	ea	gitoneac	00000001	ea	vcountlo	FFFF8209	ea
CURS_HIDE	00000000	ea	gitoneaf	00000000	ea	vcountmid	FFFF8207	ea
CURS_NOBLINK	00000003	ea	gitonebc	00000003	ea	video	0000007E	b
CURS_SETRATE	00000004	ea	gitonebf	00000002	ea	videodata	000000AA	b
CURS_SHOW	00000001	ea	gitonecc	00000005	ea	videoptr	000000A6	b
DATASZ	00000014	a	gitonecf	00000004	ea	vr	FFFFFA17	ea
DBASE	00000010	a	giwrite	FFFF8802	ea	wavelp	0000014A	t
DLEN	00000014	a	gpipl	FFFFFA01	ea	xoffset	0000006A	b
DSIZE	00000006	a	gpo	00000040	ea	xrts	00000008	ea
OTA	00000020	a	handle	00000040	b			
ENVIR	0000002C	a	hbl	0000029E	t			
FILE_ID	00000000	a	hbluec	00000120	ea			
HEADSIZE	0000001C	ea	hdv_boot	0000047A	ea			
HITPA	00000004	a	hdv_bpb	00000472	ea			
IKBD	00000004	ea	hdv_init	0000046A	ea			
LF	0000000A	ea	hdv_mediach	0000047E	ea			
LOWTPA	00000000	a	hdv_rw	00000476	ea			
MIDI	00000003	ea	hscroll	FFFF8265	ea			
MYDTA	00000020	ea	iera	FFFFFA07	ea			
PARENT	00000024	a	ierb	FFFFFA09	ea			
PRT	00000000	ea	ikbd	00000286	t			
RAMCON	00000005	ea	ikbdec	00000118	ea			
SSIZE	0000000E	a	lmra	FFFFFA13	ea			
TAB	00000009	ea	imrb	FFFFFA15	ea			
TBASE	00000008	a	ipra	FFFFFA0B	ea			
TEXTSZ	0000000C	ea	iprb	FFFFFA0D	ea			
TLEN	0000000C	a	isra	FFFFFA0F	ea			
TSIZE	00000002	a	isrb	FFFFFA11	ea			
XXX1	00000012	a	keybd	FFFFFC02	ea			
XXX2	00000016	a	keyctl	FFFFFC00	ea			
XXX3	0000001A	a	linewid	FFFF820F	ea			
XXXX	00000028	a	memcntl	00000424	ea			
md	0000049E	ea	memconf	FFFF8001	ea			
_autopath	000004CA	ea	memval2	0000043A	ea			
_bootdev	00000446	ea	memvalid	00000420	ea			
_buf1	00000482	ea	mfp	FFFFFA00	ea			
_cmdload	00000482	ea	midl	FFFFFC06	ea			
_drvbits	000004C2	ea	midictl	FFFFFC04	ea			
_dskbufp	000004C6	ea	mystack	00017C32	b			
_frclock	00000466	ea	neobuff	000000AE	b			
_fverify	00000444	ea	neofile	00000000	d			
_hz_200	0000048A	ea	noexit	00000170	t			
_membot	00000432	ea	nubls	00000454	ea			
_memtop	00000436	ea	oldikbd	00000298	t			
_nflops	000004A6	ea	oldpal	00000000	b			
_prt_cnt	000004EE	ea	oldvbl	00000280	t			
_prtabt	000004F0	ea	palette	FFFF8240	ea			
_shell_p	000004F6	ea	palmode	00000448	ea			
_sysbase	000004F2	ea	phystop	0000042E	ea			
_timr_ms	00000442	ea	prv_aux	00000512	ea			
_v_bas_ad	0000044E	ea	prv_aux0	0000050E	ea			
_vbclock	00000462	ea	prv_lst	0000050A	ea			
_vbl_list	000004CE	ea	prv_lst0	00000506	ea			
_vblqueue	00000456	ea	resvalid	00000426	ea			
abort	000001C8	t	resvector	0000042A	ea			
aer	FFFFFA03	ea	rezmode	FFFF8260	ea			
baseaddr	00000042	b	rsr	FFFFFA2B	ea			
bigbuff	00007E2E	b	sav_context	000004AE	ea			
cmdreg	00000000	ea	save_row	000004AC	ea			
colorptr	0000045A	ea	savptr	000004A2	ea			
constate	000004A8	ea	scr	FFFFFA27	ea			
conterm	00000484	ea	scr_dump	00000502	ea			
criticret	0000048A	ea	screenpt	0000045E	ea			
datareg	00000086	ea	secreg	00000084	ea			
ddr	FFFFFA05	ea	seekrate	00000440	ea			
defshiftmd	0000044A	ea	sshiftmd	0000044C	ea			
diskctl	FFFF8604	ea	start	00000000	t			
dmahi	FFFF8609	ea	stroke	00000020	ea			
dmalo	FFFF860D	ea	swv_vec	0000046E	ea			
dmamid	FFFF860B	ea	syncmode	FFFF820A	ea			


```

00000064 3F3C0000 @ Fopen a4,#0
00000068 2F0C @ move.w #$0,-(sp)
@ move.l a4,-(sp)
@ Gemdos $3d,8
0000006A 3F3C003D @ move.w #$3d,-(sp)
0000006E 4E41 @ trap #1
@ .if $8 <= 8
00000070 504F @ addq #$8,sp
@ .else
@ add.w #$8,sp
- .endif
57 -
58 00000072 4A40 @ tst d0
59 00000074 6800xxxx @ bmi abort ; IF (Error opening a file) ABORT
60 00000078 41F9xxxxxxx @ lea handlist,a0
61 0000007E 31804000 @ move d0,(a0,d4) ; Save the Handle
62 00000082 5444 @ addq #2,d4
63 00000084 887C0010 @ cmp #16,d4
64 00000088 6E00xxxx @ bgt .gotnine
@ Fseek #128,d0,#0 ; Skip NEO Header
@ move.w #$0,-(sp)
@ move.w d0,-(sp)
@ move.l #$80,-(sp)
@ Gemdos $42,10
@ move.w #$42,-(sp)
@ trap #1
@ .if $a <= 8
- addq #$a,sp
- .else
- add.w #$a,sp
65 @ .endif
66 000000A2 4A80 @ tst.l d0
67 000000A4 6800xxxx @ bmi abort ; IF (File Seek Error) ABORT
@ Fnext
@ Gemdos $4f,2
@ move.w #$4f,-(sp)
@ trap #1
@ .if $2 <= 8
@ addq #$2,sp
@ .else
- add.w #$2,sp
- .endif
68 -
69 000000B0 60AC @ bra .neoloop
70 @ .gotnine:
@ Fread d0,#128,#bigbuff
@ move.l #bigbuff,-(sp)
@ move.l #$80,-(sp)
@ move.w d0,-(sp)
@ Gemdos $3f,12
@ move.w #$3f,-(sp)
@ trap #1
@ .if $c <= 8
- addq #$c,sp
- .else
- add.w #$c,sp
71 @ .endif
72 000000CA 4A80 @ tst.l d0
73 000000CC 6800xxxx @ bmi abort ; IF (File Read Error) ABORT
74 000000D0 45F9xxxxxxx @ lea bigbuff+4,a2
75 000000D6 41F8240 @ lea palette,a0
76 000000DA 43F9xxxxxxx @ lea oldpal,a1
77 000000E0 303C000F @ move #15,d0
78 000000E4 3200 @ .ploop: move.w (a0),(a1)+ ; save old color palette
79 000000E6 30DA @ move.w (a2)+,(a0)+ ; create new color palette
80 000000E8 51C8FFFA @ dbra d0,.ploop
81 @
82 000000EC 23FCxxxxxxxxxxxxx @ move.l #bigbuff,buffptr
83 000000F6 7E00 @ moveq #0,d7 ; d7 = Row Count
84 000000F8 49F9xxxxxxx @ .rowlp: lea threebuf,a4 ; FOR (Three rows) DO
85 000000FE 4BF9xxxxxxx @ lea handlist,a5
86 0000104 DAC7 @ adda d7,a5
87 0000106 3C3C0002 @ move #2,d6 ; d5 = Column Count
@ .redlp: Fread (a5)+,#32000,a4 ; FOR (3 Files) DO Read into temp buff
@ move.l a4,-(sp)
@ move.l #$7d00,-(sp)
@ move.w (a5)+,-(sp)
@ Gemdos $3f,12
@ move.w #$3f,-(sp)
@ trap #1
@ .if $c <= 8
- addq #$c,sp
- .else
- add.w #$c,sp
88 @ .endif
89 000011E 4A80 @ tst.l d0
90 0000120 6800xxxx @ bmi abort ; IF (File Read Error) ABORT
91 0000124 08FC7000 @ adda #32000,a4
92 0000128 51CEFF00 @ dbra d6,.redlp
93 @
94 000012C 43F9xxxxxxx @ lea threebuf,a1
95 0000132 45F9xxxxxxx @ lea threebuf+32000,a2
96 0000138 47F9xxxxxxx @ lea threebuf+64000,a3

```

```

97 0000013E 2079xxxxxxx      move.l  buffptr,a0
98 00000144 3C3C00C7              move  #199,d6      ; d6 = Scan Line Count
99 00000148 3A3C0027      .linlp: move  #39,d5      ; FOR (200 Lines) DO
100 0000014C 20D9      .t1:  move.l  (a1)+,(a0)+      ; Copy a line from screen0
101 0000014E 51C0FFFC      dbra  d5,.t1
102 00000152 3A3C0027              move  #39,d5
103 00000156 20DA      .t2:  move.l  (a2)+,(a0)+      ; Copy a line from screen1
104 00000158 51C0FFFC      dbra  d5,.t2
105 0000015C 3A3C0027              move  #39,d5
106 00000160 20D8      .t3:  move.l  (a3)+,(a0)+      ; Copy a line from screen2
107 00000162 51C0FFFC      dbra  d5,.t3
108 00000166 51CEFFE0      dbra  d6,.linlp
109 0000016A 23C8xxxxxxx      move.l  a0,buffptr
110 00000170 5C47              addq   #6,d7
111 00000172 8E7C000C              cmp    #12,d7
112 00000176 6F80              ble    .rowlp
113
114 00000178 7810              moveq   #16,d4
115 0000017A 49F9xxxxxxx      lea     handlist,a4
116 00000180 3F344000      .close: move  (a4,d4),-(sp)      ; FOR (Nine files) DO Close all
                                gemdos $3e,4      ; Fclose
                                move.w  #$3e,-(sp)
                                trap     #1
                                .if $4 <= 8
                                addq    #$4,sp
                                .else
                                add.w   #$4,sp
                                .endif
117
118 0000018C 4A40              tst     d0
119 0000018E 6000xxxx      bml     abort      ; IF (Error Closing a file) ABORT
120 00000192 5544              subq    #2,d4
121 00000194 6AEA              bpl     .close
122
123 00000196 4EB9xxxxxxx      jsr     initmaus      ; Install our own mouse handler
124
125 0000019C 23F80070xxxxxxx      move.l  vblvect,oldvbl
126 000001A4 21FCxxxxxxx0000      move.l  #vbl,vblvect      ; Capture System VBlank Interrupt
127
128 ;
129 ;      Scrolling Demo loop
130 ;
131 wavelp:
                                Bconstat CON      ; Keyboard Polling
                                move.w  #CON,-(sp)
                                Blos 1,4
                                move.w  #1,-(sp)
                                trap     #13
                                .if $4 <= 8
                                addq    #$4,sp
                                .else
                                add.w   #$4,sp
                                .endif
132
133 000001B8 4A40              tst     d0
134 000001BA 6700xxxx      beq     noexit      ; IF (Keyboard Input Available) THEN
                                Bconin  CON
                                move.w  #CON,-(sp)
                                Blos 2,4
                                move.w  #2,-(sp)
                                trap     #13
                                .if $4 <= 8
                                addq    #$4,sp
                                .else
                                add.w   #$4,sp
                                .endif
135
136 000001CA 803C0003      cmp.b   #'C'-64,d0
137 000001CE 6700xxxx      beq     exit      ; CTRL-C ==> EXIT
138
139 000001D2 60D8      noexit: bra     wavelp
140
141 exit:
142 ;
143 ;      System Tear-Down
144
145 000001D4 21F9xxxxxxx0000      move.l  oldvbl,vblvect      ; Restore System VBlank Interrupt
146
147 000001D6 4EB9xxxxxxx      jsr     unmaus      ; Restore System mouse handler
148
149 000001E2 41F9xxxxxxx      lea     oldpal,a0
150 000001E8 43F80240      lea     palette,a1
151 000001EC 303C000F      move    #15,d0
152 000001F0 32D8      .unlp:  move.w  (a0)+,(a1)+
153 000001F2 51C8FFFC      dbra    d0,.unlp      ; restore old color palette
                                abort:  User      ; return to user mode
                                gemdos  $20,6
                                move.w  #20,-(sp)
                                trap     #1
                                .if $6 <= 8
                                addq    #$6,sp
                                .else
                                add.w   #$6,sp
                                .endif

```

```

000001FE 4267      @      Pterm0      - (sp)      ; return to GEMDOS
00000200 4E41      @      clr.w      trap      #1
155 00000202 4AFC      @      illegal
156
157      ;
158      ;      VBL      Vertical-Blank Interrupt Server
159
160      ;      vbl:
161 00000204 48E78080      movem.l  d0/a0, -(sp)
162
163 00000208 3039xxxxxxx      move      xmouse, d0
164 0000020E C07C000F      and      #$0f, d0
165 00000212 11C08265      move.b   d0, hscroll      ; Xpos MOD 16 = Scroll count
166 00000216 4A00      tst.b    d0
167 00000218 6600xxxx      bne      .non0      ; IF (Scrolling) THEN 4 word offset
168 0000021C 11FC00A0820F      move.b   #160, linemid
169 00000222 6000xxxx      bra      .join
170 00000226 11FC009C820F      .non0:   move.b   #156, linemid
171      .join:
172 0000022C 41F9xxxxxxx      lea      bigbuff, a0
173 00000232 3039xxxxxxx      move      ymouse, d0
174 00000238 C0FC01E0      mulu     #3*160, d0      ; Ypos * Linemid = Vertical offset
175 0000023C D1C0      adda.l   d0, a0
176 0000023E 3039xxxxxxx      move      xmouse, d0
177 00000244 E240      asr      #1, d0
178 00000246 C07CFFFB      and      #$fff8, d0      ; 8*(Xpos DIV 16) = Line offset
179 0000024A D0C0      adda     d0, a0      ; a0 = Video Base Address
180 0000024C 23C8xxxxxxx      move.l    a0, vbltemp
181 00000252 11F9xxxxxxx0000      move.b   vbltemp+1, vcounthi
182 0000025A 11F9xxxxxxx0000      move.b   vbltemp+2, vcountmid
183 00000262 11F9xxxxxxx0000      move.b   vbltemp+3, vcountlo
184
185 0000026A 4CDF0101      movem.l  (sp)+, d0/a0
186 0000026E 4EF9      .dc.w    $4ef9
187 00000270 00000000      oldvbl:  .dc.l    0      ; JMP (Old-Vblank)
188 00000274 4AFC      illegal
189
190      ;
191      ;      MOUSE HANDLING
192      ;
193      ;
194      ;
195      ;
196      ;      INITMAUS      Capture system mouse
197      ;
198      ;      Given:
199      ;          Control
200      ;
201      ;      Returns:
202      ;          With motion and button vectors captured
203      ;
204      ;      Register Usage:
205      ;          destroys d0-d3 and a0-a3
206      ;
207      ;      Externals:
208      ;          none
209      ;
210      ;      initmaus:
211 00000276 A000      .dc.w    $a000      ; Line-A Trap
212 00000278 33E8FDA6xxxxxxx      move      cur_x(a0), xmouse
213 00000280 33E8FDA8xxxxxxx      move      cur_y(a0), ymouse
214 00000288 23E8FFCExxxxxxx      move.l    movec(a0), moldvec
215 00000290 217Cxxxxxxx0000      move.l    #ourmaus, movec(a0)      ; Take over mouse motion
216 00000298 4E75      rts
217      ;
218      ;      Mouse Motion Interrupt
219      ;
220      ;      ourmaus:
221 0000029A 33C0xxxxxxx      move      d0, xmouse
222 000002A0 33C1xxxxxxx      move      d1, ymouse      ; Save new mouse position
223 000002A6 4EF9      .dc.w    $4ef9
224      moldvec:
225 000002A8 00000000      .dc.l    0      ; JMP (Old motion vector)
226 000002AC 4AFC      illegal
227      ;
228      ;      UNMAUS      Restore mouse to system
229      ;
230      ;      Given:
231      ;          Control
232      ;
233      ;      Returns:
234      ;          Mouse and button vectors restored to system
235      ;
236      ;      Register Usage:
237      ;          destroys d0-d3 and a0-a3
238      ;
239      ;      Externals:
240      ;          none
241      ;
242      ;      unmaus:

```

```

243 000002AE A000                .dc.w $a000                ; Line-A Trap
244 000002B0 2179000002A8FFCE    move.l moldvec,movec(a0)    ; Restore mouse motion
245 000002B8 4E75                rts
246
247
248
249 ;
250 ; DATA STORAGE
251 ;
252 000002BA                .data
253 neofiles:                ; NEO filename search string
254 00000000 2A2E6E656F00        .dc.b "*.neo",0
255
256                                .even
257
258 ;
259 ; RANDOM DATA STORAGE
260 ;
261 00000006                .bss
262
263 oldpal:                  .ds.l 16                ; Original color palette
264
265 handlist:                ; Array of Active Handles (9)
266 00000040 =00000009        .ds.w 9
267
268 buffptr:                ; Load ptr for bigbuff
269 00000052 =00000001        .ds.l 1
270
271 bigbuff:                ; Mega-Image Buffer
272 00000056 =00046500        .ds.b 9*32000
273
274 threebuf:               ; Temporary Triple-Image Buffer
275 00046556 =00017700        .ds.b 3*32000
276
277 vbltemp:                ; Vblank Temporary Storage
278 00050C56 =00000001        .ds.l 1
279
280 xmouse:                ; Latest mouse Xposn
281 00050C5A =00000001        .ds.w 1
282
283 ymouse:                ; Latest mouse Yposn
284 00050C5C =00000001        .ds.w 1
285
286 mystack:                ; (stack body)
287 00050C5E =00000100        .ds.l 256
288
289                                .ds.l 1                ; Local Stack Storage
290
291                                .end

```

.close	00000180	t	diskctl	FFFF8604	ea	smv_vec	0000046E	ea
.gotnine	000000B2	t	dmahi	FFFF8609	ea	syncmode	FFFF820A	ea
.join	0000022C	t	dmalo	FFFF860D	ea	tacr	FFFFFA19	ea
.linlp	00000148	t	dmamid	FFFF8608	ea	tadr	FFFFFA1F	ea
.neoloop	0000005E	t	dtr	00000010	ea	tbcr	FFFFFA18	ea
.non0	00000226	t	end_los	000004FA	ea	tldr	FFFFFA21	ea
.ploop	000000E4	t	etv_critic	00000404	ea	tcdr	FFFFFA1D	ea
.redlp	0000010A	t	etv_term	00000408	ea	tcdr	FFFFFA23	ea
.rowlp	000000F8	t	etv_timer	00000400	ea	tddr	FFFFFA25	ea
.t1	0000014C	t	etv_xtra	0000040C	ea	themd	0000048E	ea
.t2	00000156	t	exec_os	000004FE	ea	threebuf	00046556	b
.t3	00000160	t	exit	00000104	t	trkreg	00000082	ea
.unplp	000001F0	t	fifo	FFFF8606	ea	trpl4ret	00000486	ea
AUX	00000001	ea	flock	0000043E	ea	tsr	FFFFFA2D	ea
B0ASE	00000018	a	glaamp	00000008	ea	ucr	FFFFFA29	ea
BLEN	0000001C	ea	gibamp	00000009	ea	udr	FFFFFA2F	ea
BPSZ	00000100	ea	gicamp	0000000A	ea	unmaus	000002AE	t
B5IZE	0000000A	a	gicrnulp	0000000C	ea	vbasehi	FFFF8201	ea
B55SZ	0000001C	ea	gifienvlp	00000008	ea	vbaselo	FFFF820D	ea
CMOLINE	00000000	a	gimixer	00000007	ea	vbasemid	FFFF8203	ea
CON	00000002	ea	ginoise	00000006	ea	vbl	00000204	t
CR	00000000	ea	giporta	0000000E	ea	vblsem	00000452	ea
CURS_BLINK	00000002	ea	giportb	0000000F	ea	vbltemp	0005DC56	b
CURS_GETRATE	00000005	ea	giread	FFFF8800	ea	vblvect	00000070	ea
CURS_HIDE	00000000	ea	giselect	FFFF8800	ea	vcounthi	FFFF8205	ea
CURS_NOBLINK	00000003	ea	gitoneac	00000001	ea	vcountlo	FFFF8209	ea
CURS_SETRATE	00000004	ea	gitoneaf	00000000	ea	vcountmid	FFFF8207	ea
CURS_SHOW	00000001	ea	gitonebc	00000003	ea	vr	FFFFFA17	ea
DATASZ	00000014	ea	gitonebf	00000002	ea	wavelp	000001AC	t
DBASE	00000010	a	gitonecc	00000005	ea	xmouse	0005DC5A	b
OLEN	00000014	a	gitonecf	00000004	ea	xrts	00000008	ea
DSIZE	00000006	a	giwrite	FFFF8802	ea	ymouse	0005DC5C	b
DTA	00000020	a	gpi	FFFFFA01	ea			
ENVIR	0000002C	a	gpo	00000040	ea			
FILE_ID	00000000	a	handlist	00000040	b			
HEADSIZE	0000001C	ea	hdv_boot	0000047A	ea			
HITPA	00000004	a	hdv_bpb	00000472	ea			
IKBD	00000004	ea	hdv_init	0000046A	ea			
LF	0000000A	ea	hdv_mediacb	0000047E	ea			
LOHTPA	00000000	a	hdv_rw	00000476	ea			
MIDI	00000003	ea	hscroll	FFFF8265	ea			
MYDTA	00000020	ea	iera	FFFFFA07	ea			
PARENT	00000024	a	ierb	FFFFFA09	ea			
PRT	00000000	ea	imra	FFFFFA13	ea			
RAWCON	00000005	ea	imrb	FFFFFA15	ea			
SSIZE	0000000E	a	initmaus	00000276	t			
TAB	00000009	ea	ipra	FFFFFA08	ea			
TBASE	00000008	a	iprb	FFFFFA0D	ea			
TEXTSZ	0000000C	ea	isra	FFFFFA0F	ea			
TLEN	0000000C	a	isrb	FFFFFA11	ea			
TSIZE	00000002	a	keybd	FFFFFC02	ea			
XXX1	00000012	a	keyctl	FFFFFC00	ea			
XXX2	00000016	a	linemid	FFFF820F	ea			
XXX3	0000001A	a	memcntl	00000424	ea			
XXXX	00000028	a	memconf	FFFF8001	ea			
md	0000049E	ea	memval2	0000043A	ea			
_autopath	000004CA	ea	memvalid	00000420	ea			
_bootdev	00000446	ea	mfp	FFFFFA00	ea			
_buf1	000004B2	ea	midi	FFFFFC06	ea			
_cmdload	00000482	ea	midictl	FFFFFC04	ea			
_drvbits	000004C2	ea	moldvec	000002A8	t			
_dskbufp	000004C6	ea	movec	FFFFFCFE	ea			
_frclock	00000466	ea	mystack	0005E05E	b			
_fver1fy	00000444	ea	neofiles	00000000	d			
_hz_200	000004BA	ea	noexit	000001D2	t			
_membot	00000432	ea	nvbls	00000454	ea			
_memtop	00000436	ea	oldpal	00000000	b			
_nflops	000004A6	ea	oldvbl	00000270	t			
_prt_cnt	000004EE	ea	ourmaus	0000029A	t			
_prtabt	000004F0	ea	palette	FFFF8240	ea			
_shell_p	000004F6	ea	palmode	00000448	ea			
_sysbase	000004F2	ea	phystop	0000042E	ea			
_timr_ms	00000442	ea	prv_aux	00000512	ea			
_v_bas_ad	0000044E	ea	prv_auxo	0000050E	ea			
_vbclock	00000462	ea	prv_lst	0000050A	ea			
_vbl_list	000004CE	ea	prv_lsto	00000506	ea			
_vblqueue	00000456	ea	resvalid	00000426	ea			
abort	000001F6	t	resvector	0000042A	ea			
aer	FFFFFA03	ea	rezmode	FFFF8260	ea			
bigbuff	00000056	b	rsr	FFFFFA2B	ea			
buffptr	00000052	b	sav_context	000004AE	ea			
cmdreg	00000080	ea	save_rom	000004AC	ea			
colorptr	0000045A	ea	savptr	000004A2	ea			
constate	000004A8	ea	scr	FFFFFA27	ea			
conterm	00000484	ea	scr_dump	00000502	ea			
criticret	0000048A	ea	screenpt	0000045E	ea			
cur_x	FFFFFDA6	ea	secreg	00000084	ea			
cur_y	FFFFFDA8	ea	seekrate	00000440	ea			
datareg	00000086	ea	sshiftmd	0000044C	ea			
ddr	FFFFFA05	ea	start	00000000	t			
defshiftmd	0000044A	ea	stroke	00000020	ea			

STE Digitized Sound

Developer information

The Atari STE™ family of computers is equipped to reproduce digitized sound using DMA (direct memory access; that is, without using the 68000). This document provides the information required to understand and use this feature.

OVERVIEW

Sound is stored in memory as digitized samples. Each sample is a number, from -128 to +127, which represents displacement of the speaker from the "neutral" or middle position. During horizontal blanking (transparent to the processor) the DMA sound chip fetches samples from memory and provides them to a digital-to-analog converter (DAC) at one of several constant rates, programmable as (approximately) 50KHz (kilohertz), 25KHz, 12.5KHz, and 6.25KHz. This rate is called the sample frequency.

The output of the DAC is then filtered to a frequency equal to 40% of the sample frequency by a four-pole switched low-pass filter. This performs "anti-aliasing" of the sound data in a sample-frequency-sensitive way. The signal is further filtered by a two-pole fixed frequency (16kHz) low-pass filter and provided to a National LMC1992 Volume/Tone Controller. Finally, the output is available at an RCA-style output jack on the back of the computer. This can be fed into an amplifier, and then to speakers, headphones, or tape recorders.

There are two channels which behave as described above; they are intended to be used as the left and right channels of a stereo system when using the audio outputs of the machine. A monophonic mode is provided which will send the same sample data to each channel.

The stereo sound output is also mixed onto the standard ST audio output sent to the monitor's speaker. The ST's GI sound chip output can be mixed to the monitor and to both stereo output jacks as well.

DATA FORMAT

Each sample is stored as a signed eight-bit quantity, where -128 (80 hex) means full negative displacement of the speaker, and 127 (7F hex) means full positive displacement. In stereo mode, each word represents two samples: the upper byte is the sample for the left channel, and the lower byte is the sample for the right channel. In mono mode each byte is one sample. However, the samples are always fetched a word at a time, so only an even number of mono samples can be played.

A group of samples is called a "frame." A frame may be played once or can automatically be repeated forever (until stopped). A frame is described by its start and end addresses. The end address of a frame is actually the address of the first byte in memory *beyond* the frame; a frame starting at address 21100 which is 10 bytes long has an end address of 21110.

Before continuing, please familiarize yourself with the DMA sound chip register set:

REGISTER DESCRIPTIONS

```
FF8900 ---- ---- ---- --cc RW Sound DMA Control
      cc:
      00 Sound DMA disabled (reset state).
      01 Sound DMA enabled, disable at end of frame.
      11 Sound DMA enabled, repeat frame forever.

FF8902 ---- ---- 00xx xxxx RW Frame Base Address (high)
FF8904 ---- ---- xxxx xxxx RW Frame Base Address (middle)
FF8906 ---- ---- xxxx xxx0 RW Frame Base Address (low)

FF8908 ---- ---- 00xx xxxx RO Frame Address Counter (high)
FF890A ---- ---- xxxx xxxx RO Frame Address Counter (middle)
FF890C ---- ---- xxxx xxx0 RO Frame Address Counter (low)

FF890E ---- ---- 00xx xxxx RW Frame End Address (high)
FF8910 ---- ---- xxxx xxxx RW Frame End Address (middle)
FF8912 ---- ---- xxxx xxx0 RW Frame End Address (low)

FF8920 0000 0000 m000 00rr RW Sound Mode Control
      rr:
      00 6258 Hz sample rate (reset state)
      01 12517 Hz sample rate
      10 25033 Hz sample rate
      11 50066 Hz sample rate
      m:
      0 Stereo Mode (reset state)
      1 Mono Mode

FF8922 xxxx xxxx xxxx xxxx RW MICROWIRE™ Data register
FF8924 xxxx xxxx xxxx xxxx RW MICROWIRE™ Mask register
```

Note: a zero can be written to the DMA sound control register at any time to stop playback immediately.

The frame address registers occupy the low bytes of three consecutive words each. The high bytes of these words do not contain anything useful, and it is harmless to read or write them. The frame address counter register is read-only, and holds the address of the next sample word to be fetched.

PROGRAMMING CONSIDERATIONS

The simplest way to produce a sound is to assemble a frame in memory, write the start address of the frame into the Frame Start Address register, and the end address of the frame into the Frame End Address register, set the Mode register appropriately (set stereo or mono, and the sample frequency), and write a one into the Sound DMA Control register. The frame will play once, then stop.

To produce continuous sound, and link frames together, more elaborate techniques are required.

The DMA sound chip produces a signal called "DMA sound active" which is one when the chip is playing sounds, and zero when it's not. When a frame ends in the repeat mode (mode 3), there is a transition from "active" to "idle" and back again on this signal. The signal is presented as the external input to MFP Timer A. You can put Timer A into Event Count mode and use it to generate an interrupt, for example when a frame has played a given number of times. Because of the design of the MFP, the active edge for this signal must be the same as the input on GPIF I4, which is the interrupt line from the keyboard and MIDI interfaces. It is, and the Active Edge Register is already programmed for that, so you need not worry about that if you use Timer A to count frames.

The DMA Sound chip's mode 3 (repeat mode) ensures seamless linkage of frames, because the start and end registers are actually double-buffered. When you write to these registers, what you write really goes into a "holding area". The contents of the holding area go into the true registers at the end of the current frame. (Actually, they go in when the chip is idle, which means right away if the chip was idle to begin with.)

If you have two frames which you want played in succession, you can write the start and end addresses of the first frame into the chip, then set its control register to 3. The first frame will begin playing. You can then immediately write the start and end addresses of the second frame into the chip: they will be held in the holding area until the first frame finishes, then they'll be copied into the true registers and the second frame will play. The interrupt between frames will still happen, so you can tell when the first frame has finished. Then, for instance, you can write the start and end registers for the start of a *third* frame, knowing that it will begin as soon as the second frame has finished. You could even write new data into the first frame and write its start and end address into the chip; this kind of ping-pong effect is rather like double-buffering of a graphics display.

Here is an example of using Timer A in Event Count mode to play a controlled series of frames. Suppose you have three frames, A, B, and C, and you want to play frame A three times, then frame B five times, and finally frame C twice. The sequence of steps below will accomplish this. Numbered steps are carried out by your program; the bracketed descriptions are of things which are happening as a result.

1. Set Timer A to event count mode, and its counter to 2 (not 3).

2. Write Frame A's start & end addresses into the registers.
3. Write a 3 to the sound DMA control register. [Play begins.] Go do something else until interrupted.

[At the end of the second repetition of Frame A, the timer's interrupt fires. At the same time, frame A begins its third repetition.]

4. Write Frame B's start and end addresses into the DMA sound chip. These values will be held until the third repetition of Frame A finishes.
5. Set Timer A's count register to 5, then go away until interrupted

[When the current repetition finishes, the start & end registers are loaded from the holding area, and Frame B will begin playing. The end-of-frame signal will cause Timer A to count from 5 to 4. At the end of Frame B's fourth repetition, its fifth will start, the timer will count down from 1 to 0, and the interrupt will occur.]

6. Write frame C's start & end addresses into the registers, and program Timer A to count to 2. Go away until interrupted.

(When the current repetition (B's fifth) finishes, the start & end registers are loaded from the holding area, and Frame C will begin playing. The end-of-frame signal causes Timer A to count down from 2 to 1. When Frame C finishes its first repetition, Timer A counts down from 1 to 0 and interrupts.)

7. Write a 1 to the DMA Sound Control Register to play the current frame, then stop. Disable Timer A and mask its interrupt. You're done.

As you can see, you program the timer to interrupt after one repetition *less* than the number of times you want a frame to play. That is so you can set up the next frame while the DMA sound chip is playing the last repetition of the current frame. This ensures seamless linkage of frames.

INTERRUPTS WITHOUT TIMER A

Besides going to the external input signal of Timer A, the DMA-sound-active signal, true high, is exclusive-ORed with the monochrome-detect signal, and together they form the GPIIP I7 input to the M68901 MFP. The intent of this is to provide for interrupt-driven sound drivers without using up the last general-purpose timer in the MFP. It is a little trickier to use, however. For one thing, it causes the interrupt at the end of every frame, not after a specified number of frames. For another, the "interesting" edge on this signal depends on what kind of monitor you have.

On an ST, monochrome monitors ground the mono-detect signal, so when you read the bit in the MFP you get a zero. Color monitors do not ground it, so it reads as a one. When the DMA sound is idle (0), this is still the case. However, when the sound is active (1), the mono-detect signal is inverted by the XOR, so the bit in the MFP reads the opposite way. (The one place where the OS reads this bit is at VBLANK time, to see if you've changed monitors. The ROMs on any machine with DMA sound are appropriately modified, so you need not worry about this.)

If you want to use the mono-detect / DMA interrupt signal, you have to set up the active-edge register in the MFP to cause the interrupt at the right time. The interesting edge on the DMA signal is the falling edge, that is, from active to idle; this happens when a frame finishes. If you have a monochrome monitor, this edge is seen as a transition from 1 to 0 on MFP bit I7. However, with a color monitor, the edge will be seen as a transition from 0 to 1. Therefore, you have to program the MFP's active-edge register differently depending on which monitor you have. Make sure the DMA sound is idle (write a zero to the control register), then check MFP I7: if it's one, you have a color monitor, and you need to see the rising edge. If it's zero, you have a monochrome monitor, and you need to see the falling edge.

The DMA sound active signal goes from "active" to "idle" when a frame finishes. If it was playing in mode 1, it stays "idle" and the control register reverts to zero. If it was playing in mode 3, the signal goes back to "active" as the next frame begins. In this case, the signal is actually in the "idle" state for a very short time, but the MFP catches it and causes the interrupt, so don't worry.

Additional Considerations

Regardless of how you manage your interrupts, there is more you should know: the signal goes from "active" to "idle" when the DMA sound chip has *fetched* the last sample in the frame. There is a four-word FIFO in the chip, however, so it will be eight sample-times (four in stereo mode) before the sound actually finishes. If you are using mode 1, you can use this time to set up the chip with the start and end addresses of the next frame, so it will start as soon as the current one ends. However, if the interrupt should be postponed for four or eight sample-times, you could miss your chance to start the sound seamlessly. Therefore, for seamless linkage, use the pre-loading technique described above.

MICROWIRE™ Interface

The MICROWIRE™ interface provided to talk to the National LMC1992 Computer Controlled Volume / Tone Control is a general purpose MICROWIRE™ interface to allow the future addition of other MICROWIRE™ devices. For this reason, the following description of its use will make no assumptions about the device being addressed.

The MICROWIRE™ bus is a three wire serial connection and protocol designed to allow multiple devices to be individually addressed by the controller. The length of the serial data stream depends on the destination device. In general, the stream consists of N bits of address, followed by zero or more don't care bits, followed by M bits of data. The hardware interface provided consists of two 16 bit read/write registers: one data register which contains the actual bit stream to be shifted out, and one mask register which indicates which bits are valid.

Let's consider a mythical device which requires two address bits and one data bit. For this device the total bit stream is three bits (minimum). Any three bits of the register pair may be used. However, since the most significant bit is shifted first, the command will be received by the device soonest if the three most significant bits are used. Let's assume: 01 is the device's address, D is the data to be written, and X's are don't cares. Then all of the following register combinations will provide the same information to the device.

```
1110 0000 0000 0000 Mask
01DX XXXX XXXX XXXX Data
```

```
0000 0000 0000 0111 Mask
XXXX XXXX XXXX X01D Data
```

```
0000 0001 1100 0000 Mask
XXXX XXX0 1DXX XXXX Data
```

```
0000 1100 0001 0000 Mask
XXXX 01XX XXXD XXXX Data
```

```
1100 0000 0000 0001 Mask
01XX XXXX XXXX XXXD Data
```

As you can see, the address bits must be contiguous, and so must the data bits, but they don't have to be contiguous with each other.

The mask register must be written before the data register. Sending commences when the data register is written and takes approximately 16µsec. Subsequent writes to the data and mask registers are blocked until sending is complete. Reading the registers while sending is in progress will return a snapshot of the shift register shifting the data and mask out. This means that you know it is safe to send the next command when these registers (or either one) return to their original state. Note that the mask register does not need to be rewritten if it is already correct. That is, when sending a series of commands the mask register only needs to be written once.

Volume and Tone Control

The LMC1992 is used to provide volume and tone control. Before you go and find a data sheet for this part, be warned that we do not use all of its features. Commands for the features we do use are listed below.

Communication with this device is achieved using the MICROWIRE™ interface. See MICROWIRE INTERFACE the section for details. The device has a two bit address field, address = 10, and a nine bit data field. There is no way to reading the current settings.

Volume / Tone Controller Commands

Device address = 10

Data Field

011 DDD DDD Set Master Volume
000 000 -80 dB
010 100 -40 dB
101 XXX 0 dB

101 XDD DDD Set Left Channel Volume
00 000 -40 dB
01 010 -20 dB
10 1XX 0 dB

100 XDD DDD Set Right Channel Volume
00 000 -40 dB
01 010 -20 dB
10 1XX 0 dB

010 XXD DDD Set Treble
0 000 -12 dB
0 110 0 dB (Flat)
1 100 +12 dB

001 XXD DDD Set Bass
0 000 -12 dB
0 110 0 dB (Flat)
1 100 +12 dB

000 XXX XDD Set Mix
00 -12 dB
01 Mix GI sound chip output
10 Do not mix GI sound chip output
11 reserved

Note: The volume controls attenuate in 2 dB steps. The tone controls attenuate in 2 dB steps at 50 Hz and 15 kHz (Note: These frequencies may change).

Using the MICROWIRE™ Interface and the Volume/Tone Control Chip

The MICROWIRE™ interface is not hard to use: once you get it right, you'll never have to figure it out again.

The easiest way to use it is to ignore the flexibility, and just use one form for all commands. Since the Volume/Tone chip is the only device, and it has a total of 11 bits of address and data, your mask should be \$07ff. If you're picky, you can use \$ffe0, because the high-order bits are shifted out first, but it adds conceptual complexity. With a mask of \$07ff, the lower 9 bits of the data register are used for the data, and the next higher two bits are for the address:

```
Mask:      %0000 0111 1111 1111
Data:      %xxxx x10d dddd dddd
```

Replace the d's with the command code and its data. For example, this combination sets the master volume to \$14:

```
Mask:      %0000 0111 1111 1111
Data:      %xxxx x100 1101 0100
```

The other important concept you must understand is that the bits shift out of these registers as soon as you write the data, and it takes an appreciable time (16 µsec) to finish. You can't attempt another write until the first one is finished. If you read either register while it's being shifted out, you will see a "snapshot" of the data being shifted. You know the shifting is complete when the mask returns to its original value. (This theory is wrong if you use a mask which equals its original value sometime during the shifting, but \$07ff never does.)

Assuming you write \$07ff into the mask register ahead of time, the following routine can be used to write new data from the D0 register to the volume/tone control chip:

```
MWMASK      equ    $ffff8924
MWDATA      equ    $ffff8922

mwwrite:
    cmp.w    #$07ff,MWMASK    ; wait for prev to finish
    bne.s    mwwrite          ; loop until equal
    move.w   d0,MWDATA        ; write the data
    rts                               ; and return
```

The purpose of the loop at the beginning is to wait until a previous write completes. This loop is at the beginning of the routine, not the end, because waiting at the end would always force at 16 µsec delay, even if it's been longer than that since the last write.

